SPECIFICATION

Please amend paragraph [0057] of the Specification as follows:

In step 408, the signature in the authorization message and the object are separately sent to the set top box 108 over the network 208. Preferably an asymmetric signature is used (e.g., RSA, DSA or ECC based), but a symmetric signature (e.g., DES or triple-DES) could also be used. Upon receipt of the signature and object, the signature is correlated to the object in step 416. ~~Upon receipt of the signature and the object and before storing the object, the signature is calculated and checked by the ACP 240 in steps 420 and 424.~~ After correlating the signature and the object and before storing the object, the signature is calculated and checked by the ACP 240 in steps 420 and 424. If the calculated and received signatures match, the object is stored in step 428. Alternatively, the object is discarded in step 432 if there is no match, and processing loops back to step 412 to wait for another copy of the object.

Please amend paragraph [0067] of the Specification as follows:

Referring next to FIG. 8, an embodiment of a "rights" message 800 is shown in block diagram form. The rights message 800 conveys rights to use a functional unit. The functional unit could be an object or a resource. Typically, there is one rights message 800 for each set top box 108, which specifies any rights for all functional units. Requirements from the authorization message 500 that are associated with objects and resources are checked against the rights to determine if interaction with another object or resource is authorized. The rights message 800 allows remotely adding new rights to a functional unit associated with the set top box 108. Although not shown, the rights

message 800 typically includes a digital signature to verify the integrity of the message 800 during transport. In some embodiments, a checksum 812 could be used instead of a digital signature.

Please amend paragraph [0079] of the Specification as follows:

Once both object and authorization messages 600, 500 are loaded, all the components of the signatory group 700 are available to the ACP 240. In step 1016, the ACP 240 calculates the signature over the signatory group 700. The ACP 240 makes a determination in step 1024 as to whether the signature 512 in the authorization message 500 matches the calculated signature. If there is a match, the object 608 is authorized and the object 608 is loaded into memory 228 by the OS and allowed to execute in step 1028. Alternatively, the ACP 240 discards the object 608 and notifies the OS of an error if the signatures do not match in step 1032. A signature 512 mismatch could result from corruption during storage, a pirate replacing the object 608 or a virus corrupting the object 608.

Please amend paragraph [0084] of the Specification as follows:

Once the ACP 240 has the authorization message 500, the entitlement information therein is checked in step 1212. A determination is made in step 1216 as to whether the object 608 is authorized by checking the entitlement information. If the object 608 is authorized, it is loaded into memory by the OS and executed in step 1220. Alternatively, the OS is notified of a failed authorization attempt and object 608 is discarded in step 1224 if there is no entitlement to use the object 608.

Please amend paragraph [0089] of the Specification as follows:

A determination is made in step 1320 as to whether the authentication and authorization performed in step 1316 were both performed successfully. If successful, the process loops back to step 1312 where the process waits for the next checkpoint. Alternatively, in step 1324, the object is removed from memory 228 and discarded when either the authorization or authentication checks fail. Preferably, the ACP 240 is the time source for determining the scheduled checkpoints. The ACP 240 is less susceptible to attacks that set the clock back to avoid expiration of authorization. Additionally, the ACP 240 does not run application software that could change the time and requires secure commands to change the time. Secure commands could use encryption or signatures to guarantee authenticity of any time changes. To expire authorization, keys used for decryption could be expired or a new rights message 800 could be sent that overwrites and removes the right to use an object.

Please amend paragraph [0100] of the Specification as follows:

In step 1502, a reportback timer is set to an initial value of one day. After setting, the reportback timer starts counting down in time. For the set top box 108 subjected to this process, the headend 104 monitors for any reportback from the set top box 108 in step 1506. In step 1510, a test is performed to determine if the security report has been received. If the report is received, processing continues to step 1546 where the report is analyzed for any identified security problems. Where there are security problems, the set

top box [[1518]] 108 may be disabled in step 1518. Where there are no security problems, processing loops back to step 1502.

Please amend paragraph [0117] of the Specification as follows:

In step 1650, the plaintext remainder of the object 608 is received by the set top box 108 and stored in memory 228 in step 1654. Nothing is done to the plaintext remainder unless the user purchases use of it in step 1658. The purchase is reported back to the headend 104 by way of the network 208 in step 1662. Once any verification is performed upon the purchase request, the headend 104 sends the missing plaintext portion that is received in step 1666. A secure channel is used to send the plaintext portion to the set top box 108 that purchased the object 608.

Please amend paragraph [0118] of the Specification as follows:

In step 1670, the plaintext portion and remainder are joined to reformulate the object 608 at the set top box 108. This embodiment further requires in step 1674 that a new rights message 800 be received from the headend 104 to enable use of the object. The new rights message 800 would replace the old rights message 800 and provide rights to use the object 608.

Please amend paragraph [0119] of the Specification as follows:

With reference to FIG. 17, some of the functional units of a set top box 108 are shown. Functional units toward the bottom of FIG. 17 are superordinate to the functional units near the top of FIG. 17. That is to say, functional units toward the top of FIG. 17 are subordinate to those lower in the figure. Superordinate functional units are responsible

for imposing checkpoints on subordinate functional units. For example, the hardware 1704 imposes checkpoints upon the BIOS 1708, OS 1712 and so on up the subordination hierarchy. The BIOS 1708 imposes checkpoints on the OS 1712, but not upon the hardware 1704. In another example, Java Virtual Machine 1720 imposes checkpoints on Java Applications 1724 but not on OS 1712. Functional units in the same ordination stratum can impose a checkpoint on another functional unit in that stratum when they interact. For example, an application 1716 can require execution of a checkpoint on a driver 1718. Also, by way of example, an application 1716 may require execution of a checkpoint on a resource 1714 or Java Virtual Machine 1720.